

SPECIFICATION AMENDMENTS

Please amend the paragraph beginning on page 11, line 1, as follows:

The *condition* of a policy rule is a sequence of predicates of the form $t_1 t_2 t'$, where t and t' can be attributes from primitive events that appear in the event part of the rule, or they could be constants or the result of operations applied to the attributes of the primitive events that appear in the event. $_2$ is a comparison operator such as $<, =, >$, etc. There is a special class of operators that can be used to form terms, called aggregators. For a given "generic" aggregator Agg , the syntax of the operator will be $Agg(e.x, e.x_2 t)$ or $Agg(e)$. Here, e is a primitive or policy-defined event that appears in the event part of the proposition, $e.x$ is an attribute of e . As the name suggests, the aggregator operators are used to aggregate the x values over multiple epochs if $e.x_2 t$ holds. For example, a count aggregator $Count(e.x, e.x < 20)$ can be used to count the number of occurrences of event e for which $e.x < 20$ over multiple epochs. An aggregator could also add the values of the attribute $e.x$ or get the largest value, etc. These aggregation terms may be applied to primitive events that appear under the scope of a caret "^" operator. The rule " $e_1 \wedge e_2$ causes a if $Count(e_2) = 20$ " will execute action "a" if 20 instances of e_2 follow an instance of e_1 .

Please amend the paragraph beginning on page 23, line 3, as follows:

As shown in Fig. 12, for the embodiment of ~~Fig. 1~~ Fig. 2, the policy server 8 or a policy agent 8a receives an event in step 140. As shown in Fig. 6, the event is received at the policy server's or policy agent's policy evaluator 50. In step 142, the policy evaluator 50 performs policy evaluation. Because this procedure may depend on previous actions and/or conditions occurring at a network element that is remote from the policy server 8 or policy agent 8a, the action/condition handler 54 may be used to query the directory server 16 in step 144 to determine where the action/condition can be checked, and how it can be checked (i.e., the protocol to use

for obtaining the information). The directory server 16 performs the action/condition lookup in step 146 and reports back to the action/condition handler 54, which in turn reports to the policy evaluator 50. In step 148, the policy evaluator 50 completes policy processing and generates an action command. After the action command is generated by the policy evaluator 50, the action distribution component 56 requests domain resolution from the directory server 16 in step 150. In step 152, the directory server 16 performs the domain lookup and reports the PEP address information to the action distribution component 56. In step 154, the action distribution component 56 distributes the action command for execution at one or more recipient PEPs.

Please amend the paragraph beginning on page 24, line 15, as follows:

Fig. 13 is a functional block diagram showing the architecture of a software switch system 160. The system can be viewed as comprising a set of software components that reside on a single hardware platform or which can be distributed across multiple geographically separated hardware platforms. These components include one or more call coordinators (e.g., 162 and 164), device servers (e.g., 166, 168, 170 and 172), directory coordinators (e.g., 174), service provider servlets (e.g., 176), and user feature applets (e.g., 178). In addition, the software switch system 160 implements a policy server 180 in accordance with the present invention, e.g., according to the described construction of the policy server 8. Note that PEPs of the type described relative to Fig. 1 can be loaded into the device servers 166, 168, 170 and 172 to complete the incorporation of policy manager functionality within the software switch system 160 in accordance with the first embodiment of the invention. Alternatively, PEPs of the type described relative to Fig. 2 (i.e., containing policy agents 8a) can be loaded into the device servers 166, 168, 170 and 172 to complete the incorporation of policy manager functionality

within the software switch system 160 in accordance with the second embodiment of the invention.

Please amend the paragraph beginning on page 32, line 7, as follows:

Turning now to Fig. 17, a policy execution space 280 is shown in combination with a debugging tool 282 that is configured to help users test and debug their policies. Through a GUI 284 function provided at the user interface 17, the debugging tool 282 allows users to ask hypothetical questions about a policy. Queries may be of the form: "Give me an event history that will trigger this action," or "Complete this event history until this action is triggered," or "Find an event history that triggers a given sequence of actions." Note that the debugging tool 282 could rely on a database of pre-written hypothetical scenarios, or a database representing a history of actual network management operations.